JOINT INSTITUTE FOR NUCLEAR RESEARCH
Dzhelepov Laboratory of Nuclear Problems

FINAL REPORT ON THE START PROGRAMME

Integration of the Zero Degree Calorimeter in the SpdRoot framework

Supervisor:
Dr. Katherin Shtejer Díaz

Student:
BSc. Alexs Manuel Gaute Alvarez, Havana
Higher Institute of Technology and Applied Sciences

Participation period:
Sep 18 - Nov 9,
Summer session

Dubna, 2025

Abstract

The successful integration of the Zero Degree Calorimeter (ZDC) for the Spin Physics Detector (SPD) experiment into the SpdRoot simulation framework is presented. This work involved updating the ZDC geometry for Phase II and implementing it using ROOT's TGeo package. The sampling calorimeter comprises electromagnetic and hadronic sections, with sensitive volumes defined and new methods developed to manage geometric parameters and interaction points.

Preliminary digitization was proposed using an algorithm incorporating the detector's energy and time resolution response from Geant4 studies and parameters measured with a simplified prototype using 3 cm cubic scintillators. Simulation results confirm correct geometry implementation, with detector responses properly recorded and processed. This integration provides an essential tool for future SPD simulations, enabling realistic performance studies and physics analyses with the SPD-ZDC subsystem within the NICA collider environment.

Keywords: SpdRoot, ZDC, Monte Carlo, geometry, points, hits.

Contents

1 Introducción

1.1 SPD experiment

The Spin Physics Detector (SPD) collaboration proposes to install a universal detector in the second interaction point of the NICA collider under construction (JINR, Dubna) to study the spin structure of the proton and deuteron and other spin-related phenomena using a unique possibility to operate with polarized proton and deuteron beams at a collision energy up to 27 GeV and a luminosity up to $10^{32}\ cm^{-2}\ s^{-1}$. The main goal of the experiment is to investigate crucial issues of the nucleon spin physics. Polarized and unpolarized physics is possible. Since the SPD energy range is relatively new for spin investigations, there is a lack of polarization data, which leads us to develop methods for local polarimetry [1].

1.1.1 ZDC at SPD

A Zero Degree Calorimeter (ZDC) is a standard device for the collider environment. It is placed in the space between two dipole magnet, about 13 m from the interaction point (Fig. 1).

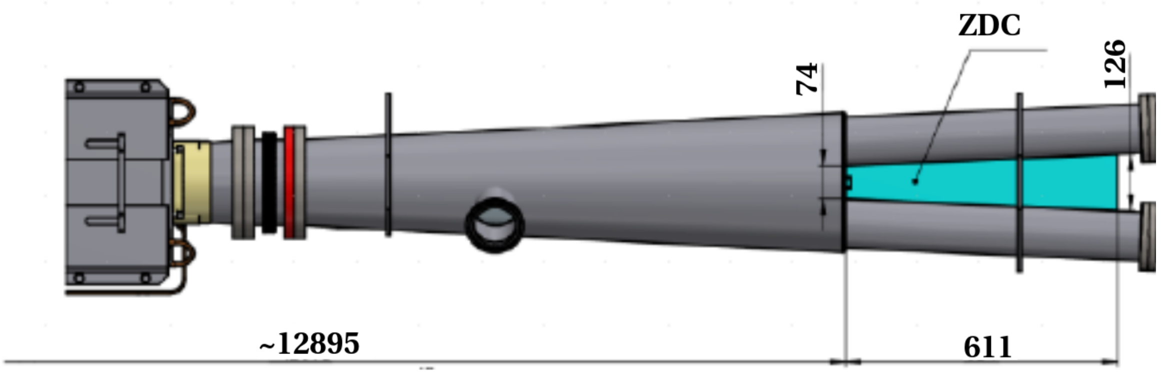


Figure 1: Dimensions and positioning of the ZDC between the beam pipes

The strong magnetic field before the ZDC efficiently removes all charged particles, allowing clean measurement of neutrals, so the device can work up to very high luminosities. Two ZDC devices are suposed to be placed symmetrically on both sides of the IP. A coincidence between them, as well as with other detectors, will be used.

1.2 Motivation of this work

The simulation framework of the SPD experiment is SpdRoot [3]. It is derived from the FairRoot software, which provides comprehensive tools for Monte Carlo simulation, event reconstruction, data analysis and visualization.

The SpdRoot software contains detailed geometrical models of all detector subsystems of SPD. However, the classes intended to describe the ZDC detector had not been updated for a long time since the first developments were made in SpdRoot and the geometry no longer corresponded to the specifications described in the Technical Design Report [1]. Furthermore, the mechanisms for recording and accounting for hit production were in their most incipient phase. This motivated the present work, aimed at updating the geometric model of ZDC and creating the conditions to develop simulation, reconstruction and physics analysis.

The main objectives are the following:

- Implement the ZDC geometry corresponding to the SPD Phase-II within the SpdRoot framework.

- Introduce additional parameters for storage during the *points* simulation for Monte Carlo events.
- Create the necessary tools for hit creation and processing starting from the simulated *points.*
- Prepare preliminary classes to enable future digitization of hits in ZDC.

2 SpdRoot framework

SpdRoot is derived from the FairRoot software, and makes posible the Monte Carlo simulation, event reconstruction, data analysis and visualization. It has provides a flexible description of subsystems and is based on the ROOT geometry package. Several generators are implemented in SpdRoot, like Pythia8 for proton-proton collisions, the FRITIOF model for deuteron-deuteron and UsQMD for nucleus-nucleus interactions. The simulation stage makes use of Geant4 to transport particles through the detector geometry.

SpdRoot encompasses classes for each active volume of the SPD experiment. For example, *SpdBbc.h*, *SpdEcalTB.h*, *SpdZdc.h*, for BBC, ECAL and ZDC respectively. These classes construct the geometric characteristics of each detector and include methods to process particle interactions in the defined sensitive volumes. Geometries are built using the tools provided by ROOT to create different types of volumes and materials. Methods for accessing and storing information related to particle interactions with the sensitive volumes are contained in clases with the following name convention: *SpdBbcPoint.h*, *SpdEcalTBPoint.h*, and *SpdZdcPoint.h*, and so on. In adition there are classes responsible for providing information about the geometry dimensions and materials of each detector, allowing users to modify them in a relatively easy way. Some of these classes include, for instance, *SpdBbcGeoMapper.h*, *SpdEcalBGeoMapper.h*, and *SpdZdcGeoMapper*.

The flux diagram of a simulation in SpdRoot is shown in the appendix B. The process begins with the event generation, where the physics involved in the collision and other beam parameters are defined. Then, the transportation and particle interactions through the detector components are described with Geant4. The output files from the simulation step, contain a tree with data about interactions with sensitive volumes, Monte Carlo information and geometry parameters, which are transferred to the reconstruction phase. Finally the processing of *points* and the event reconstruction is performed. In the SpdRoot convention, *points* are equivalent to the deposition of energy at a given point, constituting hits at the Monte Carlo level. In the final phase of the simulation (hit processing and reconstruction), other SpdRoot classes come into play. Generic classes are created for all detectors allowing to store simulated data in arrays (similar to how *points* are stored). Examples of those generic classes are *SpdHit.h* and *SpdHitMCTruth.h* In the final phase of the simulation (digitization and reconstruction), other SpdRoot classes come into play. The data are stored in arrays (similar to how *points* are stored), for which generic classes are created for all detectors, such as *SpdHit.h* and *SpdHitMC-Truth.h*, along with more detector-specific classes that inherit from these. Among these specific classes are *SpdBbcMCHit.h*, *SpdEcalTB2MCHit.h*, and *SpdZdcMCHit.h.* Specific classes of each detector, such as, such as *SpdZdcMCHitProducer.h* inherit from the generic ones.

3 ZDC description at SpdRoot

3.1 Geometry

The ZDC is a sampling calorimeter with an absorber/scintillator sandwich configuration. A depiction of the ZDC geometry for the second construction phase can be seen in figure 2.

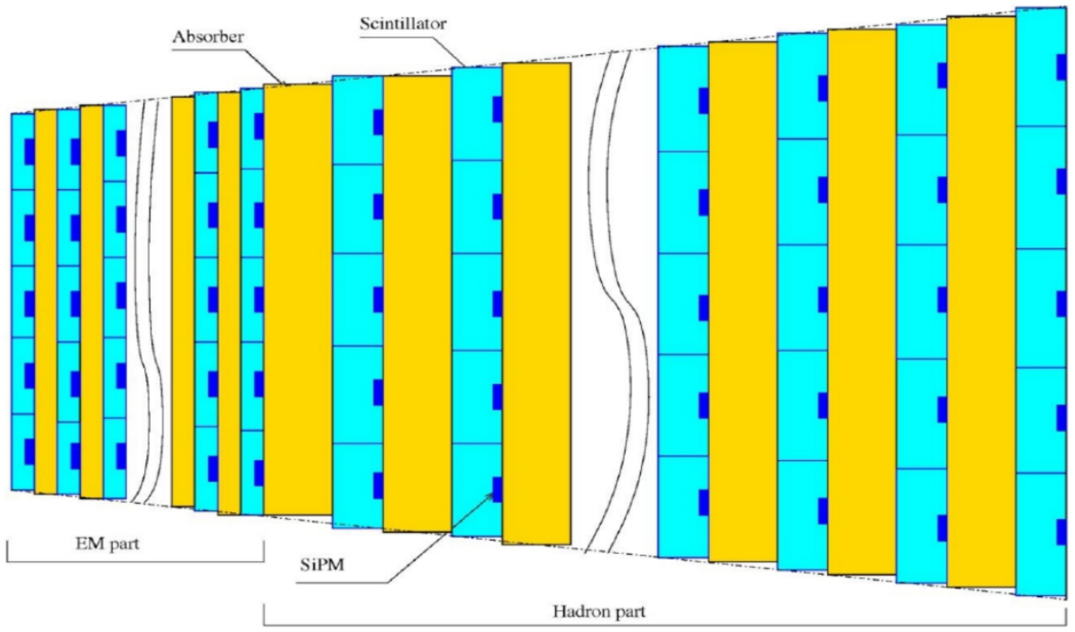


Figure 2: Representation of the ZDC in the second phase of SPD construction

As shown in the figure, the geometry consists of two fundamental sections: an electromagnetic section and a hadronic section. The electromagnetic section is composed of 8 plates, the first of which acts as a veto layer designed to detect any charged particles reaching the detector, since its primary function is the detection of photons and neutrons. The hadronic section, located directly behind the electromagnetic one, comprises 22 plates, resulting in a total of 30 plates that constitute the entire detector volume. Each plate is composed of an absorber, a 5×7 scintillator matrix, and a PCB with photomultipliers embedded within each scintillator tile. The dimensions of the individual components for each calorimeter section and their respective materials are provided in Tables 1 and 2.

Table 1: Geometric parameters

Parameter	Value
SiPM size, mm	3
PCB thickness, mm	1
Electromagnetic part	
Number of layers	8
Sci. thickness, mm	5
Abs. thickness, mm	5
Hadron part	
Number of layers	22
Sci. thickness, mm	10
Abs. thickness, mm	13

Table 2: Material composition

Component	Material
Absorber	Tungsten
PCB	G10
Scintillator	Polystyrene
SiPM	S13360-3050PE

The description of geometries in SpdRoot is accomplished using ROOT's geometry constructor, *TGeo*. The first step involves defining the media for each volume; for this purpose, tungsten and G10, already defined in SpdRoot's *media.geo*, were utilized. The *TGeoMedium* for the scintillators and the photomultipliers were defined from *TGeoMixture* instances, which were created based on their constituent *TGeoElement*s.

For constructing the shapes of each detector component, a simple inspection of figure 2 reveals that the volumes have an orthohedral (rectangular prism) shape. The ROOT manager, *TGeoManager*, provides the *MakeBox* method to create such volumes. For the scintillator tiles, *TGeoCompositeShape* a Boolean volume enabling operations like union, subtraction, intersection, and difference between two or more volumes—was employed. Since the SiPMs are located inside the scintillators, the SiPM volume was subtracted from the scintillator volume, translated along the X-axis to align with the PCB plate behind the scintillators.

The transverse dimensions of the geometry are 14.3 cm along the X-axis. As observed in figure 2, the Y-dimension of the detector increases linearly. This presented a minor challenge for geometry construction due to the increased number of logical volumes required. A method was implemented to return the Y-size depending on the Z-coordinate of the volume center. It was also necessary to create *TGeoVolume** pointers to reserve memory space for each volume to be built. A ROOT tool, *TGeoVolumeAssembly*, was leveraged to improve geometry modularization. Briefly explained, this tool is a volume that does not require a material medium and can host other volumes, acting as a mother volume with its own coordinate system. A portion of the code illustrating some details of the methods used is provided in Appendix C.

After creating all the required volumes, they were positioned using the *AddNode* method, as represented in Listing 1.

```
MotherVolume -> AddNode(TGeoVolume *DuaghterVolume, Int_t CopyNo,TGeoMatrix
    *mtx);
```

Listing 1: Method for locating volumes in ROOT geometries

The copy numbers for the scintillators arranged in a 5 × 7 matrix within each layer were defined as: $copyNo = (\text{row}) \cdot 10 + (\text{column}) \cdot 1$. For example, a $copyNo = 34$ corresponds to the scintillator in row 3 and column 4. In ROOT, geometry names are defined according to the hierarchical order of the geometry, as illustrated in Appendix D. An example of such a name for the scintillator located in row 3, column 1 of layer number 15, in the hadronic part of the negative ZDC module, is: $World\_1/module2\_1/HadronicPart2\_1/Layer15\_15/scihole15\_31$.

All the aforementioned work was carried out in the method SpdZdc::ConstructDetector(), which enables the description of detector geometries within the SpdRoot framework. Within this method, the scintillators were also defined as sensitive volumes using the method *FairModule::AddSensitiveVolume*.

Additionally, new methods were implemented in the SpdGeoMapper class to allow users to access and modify geometry parameters, such as the distance between modules (Listing 2), the absorber thickness, or the scintillator thickness. The newly created methods are shown in Appendix E.

```
 SpdZdcGeoMapper* mapper = SpdZdcGeoMapper::Instance(); //Instance for
    GeoMapper
SpdParameter* par = mapper->AddParameter("ZdcMinDist"); //pointer to
    ZdcMinDist where it is storage the distance between both modules
*par = 50.; //modify the parameter for 50 cm
```

Listing 2: Example of how to modify the distance between the ZDC modules

3.2 Storing *points*

Points serve as the bridge between the Geant4 simulation and the subsequent data analysis. They contain the "raw" physical information about what actually occurred within the detector during the simulation. These are objects that store data on particle interactions with sensitive volumes. For the ZDC, a specific class named *SpdZdcPoint* exists, which inherits from *FairMCPoint*. This class provides methods for storing information into collections and for accessing it via getter methods. Due to the geometry modifications and the definition of new sensitive volumes, this class was adapted to also record the layer number and the specific scintillator tile where the interaction took place.

The interaction data are collected in the *SpdZdc::ProcessHits* method. This is a standard Geant4 method that tracks a particle step-by-step as it traverses a volume declared as sensitive. Within this method, data such as the particle's momentum, its arrival time, and the energy deposited at each step are retrieved. In our implementation, the path of the sensitive volumes was accessed, and the copy number of a volume was obtained using *SpdGeopathParser::Num*. The following listing illustrates this implementation.

```
    SpdGeopathParser parser;
    parser.ParsePath(fVolumePath);
    fModuleID = parser.Num(2,true); //moduleID=> z+ is 1; z- is 2
```

Listing 3: SpdGeopathParser example

Once all the variables to be saved in the points are updated, the *SpdZdc::AddHit* method is called. This method invokes the constructor of the *SpdZdcPoint* class to add the new point to a *TClonesArray*. The updated source code f or the aforementioned classes can be found in Appendix F. In summary, the simulation flow at this stage is as follows:

$$\text{Particle} \rightarrow \text{Interaction}(SD) \rightarrow ProcessHits() \rightarrow \text{Point creation} \rightarrow \text{Storage at } TClonesArray$$

3.3 Hit production

As detailed in Appendix B, the next step after the detector response is the digitization of the *points*, which are then referred to as hits. Before describing the methods used in the digitization, it is necessary—just as with the points—to construct a class containing the methods for storing and accessing the information of these hits. For the ZDC, the class *SpdZdcMCHit* exists, which inherits from *SpdHit* and *SpdHitMCTruth*, classes that provide general methods for the other detectors. The methods implemented in the *SpdZdcMCHit* class can be seen in Appendix G. The newly created methods are again related to the geometry modification and the need to know in which layer and scintillator tile the interaction occurred for any type of physical analysis.

Digitization is implemented in the class *SpdZdcMCHitProducer*, which inherits from several classes, including, *SpdZdcMCHit* and *SpdZdcPoint*. For the digitization, information reported by the team responsible for ZDC construction was used, and the energy resolution for each detector section, reported in a previous work [2] was also included. According to the experimental calibration performed with muons and 5 mm scintillator tiles, the following relationship was obtained:

$$\frac{\Delta E}{5mm} \approx 20phe/MeV \tag{1}$$

This value of photoelectrons per MeV of deposited energy was stimated for the electromagnetic section. To determine the relationship for the scintillators in the hadronic part, a linear relationship between the number of photoelectrons and the tile dimensions was assumed, as follows:

$$\frac{\Delta E}{10mm} \approx 40phe/MeV \tag{2}$$

The energy resolutions described are the following:

$$RMS_{electr} = \frac{0.19}{\sqrt{E[MeV]}} \tag{3}$$

$$RMS_{had} = \frac{0.52}{\sqrt{E[MeV]}} \tag{4}$$

Finally, the following relationship for the time resolution, also a result of the detector calibration, was used:

$$RMS_{time} = \frac{0.8ns}{\sqrt{E[MeV]}} \tag{5}$$

For the digitization a Gaussian smearing was applied to the energy deposited by the particle in the scintillator. The smearing has a mean value equal to the deposited energy, and the value of σ is obtained by substituting this energy into equations 3 and 4. Using the result of the previous smearing, the number of photoelectrons equivalent to this calibrated energy is found using equation 1 or 2. To determine the number of detected photoelectrons, a Gaussian smearing is performed with a mean value equal to the number of photoelectrons equivalent to the calibrated energy and $\sigma = \sqrt{\langle nPhe \rangle}$. Finally, after obtaining the number of photoelectrons from the smearing, the conversion back to energy is performed, and this value is the output of the digitization. For the time resolution, the energy is introduced in equation 5, and this value is passed as an argument to the method *FairTimeStamp::SetTimeStampError*, which stores the time resolution for each hit. The methods created in this class can be observed in Appendix G.

4 Results

As a result the figure 3 shows the two ZDC modules constructed in ROOT. These modules are positioned 10 *cm* apart from each other to facilitate visualization of the volumes; subsequently, in *SpdZdc.h*, they were placed at 1289.5 *cm* from the interaction point.

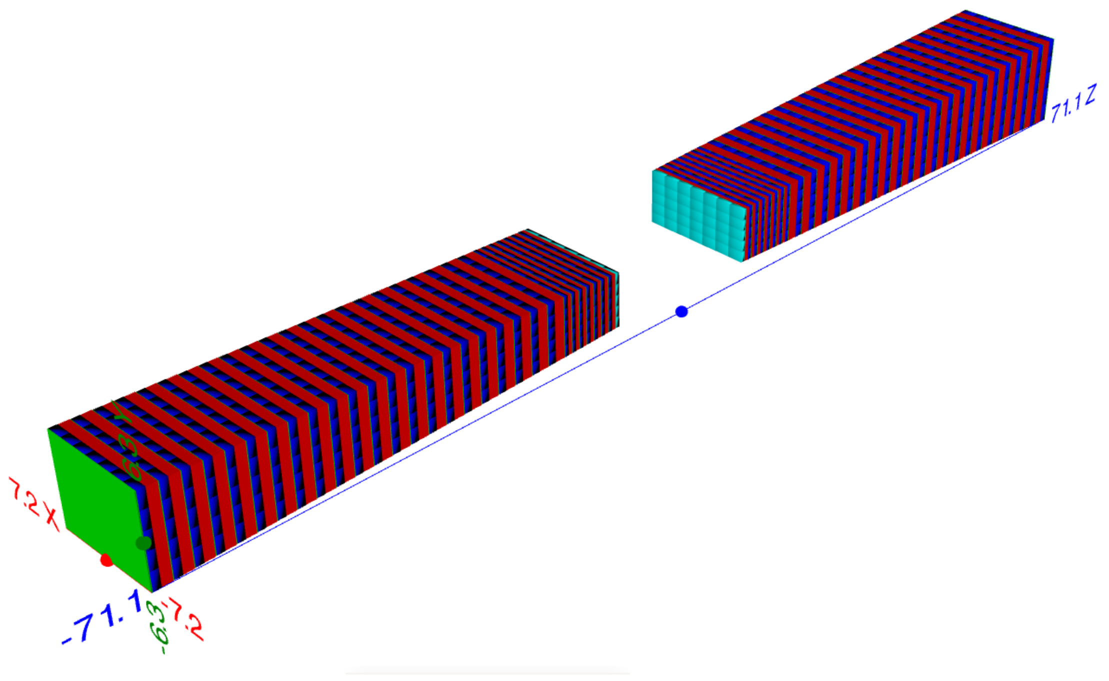


Figure 3: ROOT representation of the ZDC phase 2 geometry. Red color indicates the absorber, green represents the PCB, cyan is used exclusively for the veto scintillators, and blue for the remaining scintillators

Figure 4 displays the two modules with a transverse cut ($x > 0$), revealing the photomultipliers inside the scintillators and the internal structure of the detector.

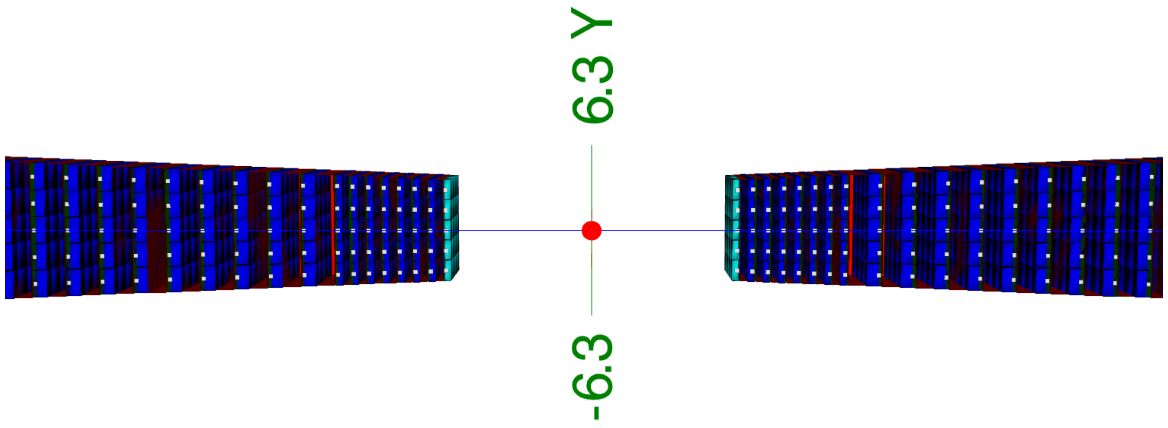


Figure 4: Transverse cut of the ZDC modules

Finally, the following figure presents a rear view of the detector without the PCB. The 5×7 scintillator matrix is visible, consistent with the design reported in [1]. The numbering scheme used for the scintillator copy numbers, as described in section 3.1, is also shown.

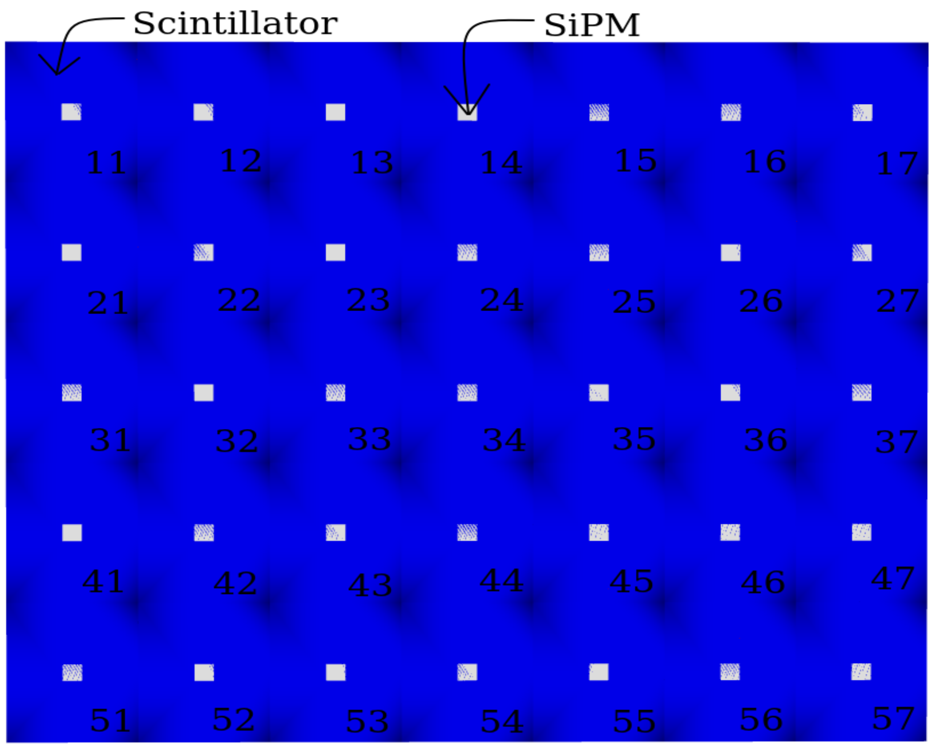


Figure 5: Rear view of one module showing scintillators in blue with white SiPMs inside

To verify the correct definition of the scintillators as sensitive volumes, a simulation was performed using the macro described in Appendix B (*simu.C*). The simulation consisted of 1000 events and included only the ZDC detector, with the magnetic field disabled. The output data from this macro were analyzed, and some of the results are shown in the following figures (Fig. 6, 7, 8, 9).

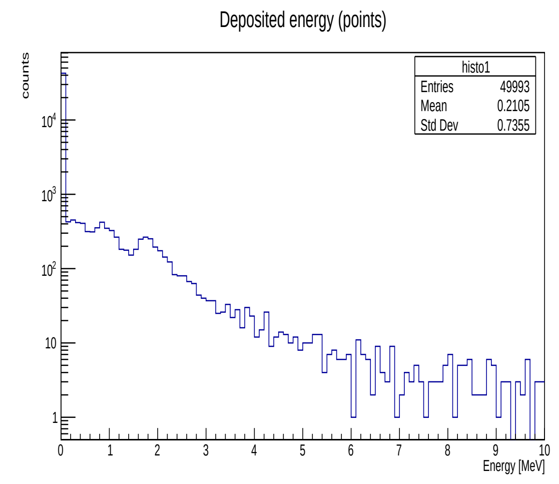


Figure 6: Deposited energy per point

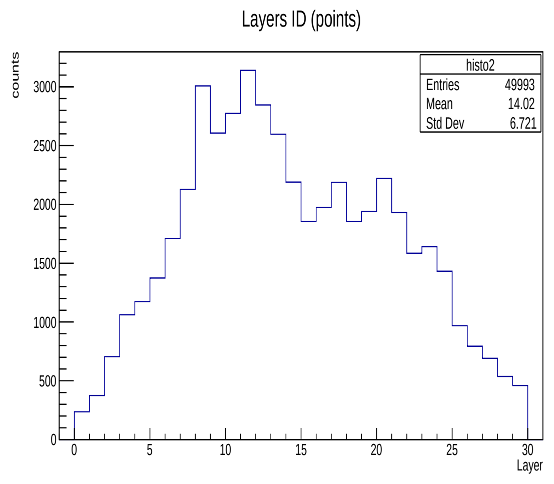


Figure 7: Number of points per layer

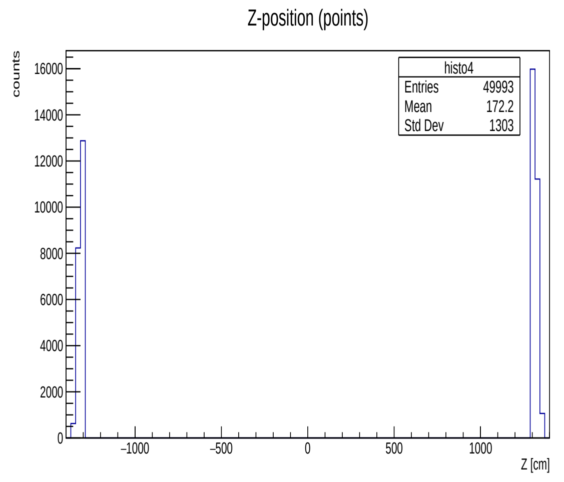


Figure 8: Z-position of points

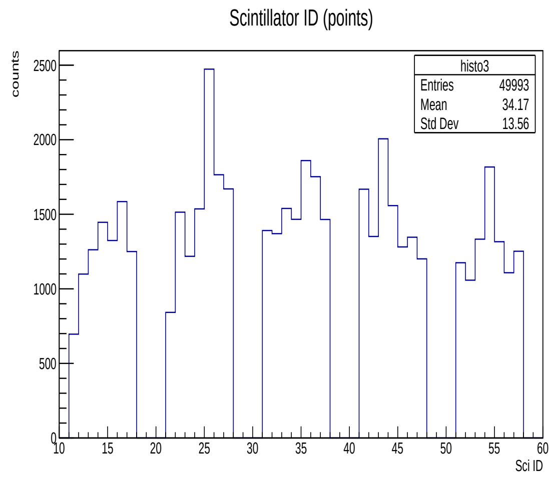


Figure 9: Points per scintillator copy number

As observed in these four figures, the sensitive volumes were correctly recognized by SpdRoot. Figure 6 shows a histogram of the energy deposited by *points*, which includes both, particles that deposited energy in the scintillators and those that traversed without energy loss. Figure 7 displays the number of *points* per detector layer, while figure 8 shows their distribution along the Z-position, clearly indicating the locations of the two modules within the SPD coordinate system. The final figure (9) is particularly illustrative, as it shows the number of *points* per scintillator tile organized according to their copy number, matching the scheme presented in figure 5.

Following the procedure outlined in Appendix B, we proceeded with the reconstruction using the *reco.C* macro on the data obtained from the *simu.C* simulation. Selected reconstruction results are presented in figures 11, 13, 15, and 17, on the rigth column. For comparison with the point data, when no smearing was performed, events with zero deposited energy were filtered out, and the corresponding plots (10, 12, 14, and 16) were generated. The subsequent figures enable a direct comparison between these datasets.

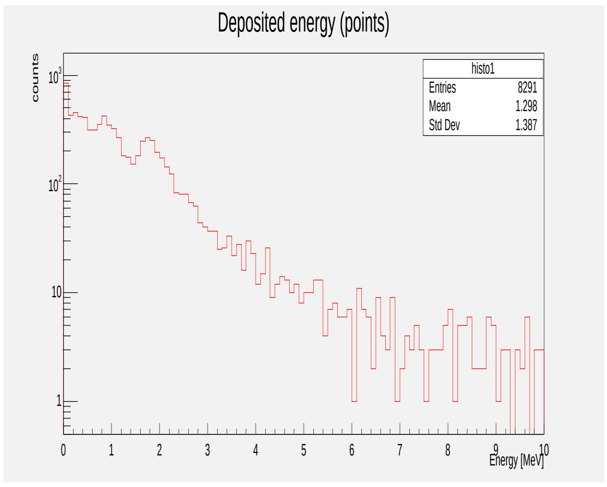


Figure 10: Deposited Energy $\neq 0$ for points

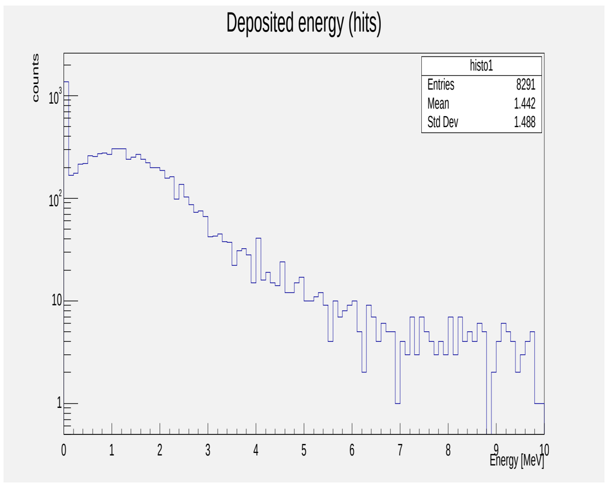


Figure 11: Deposited energy by hit

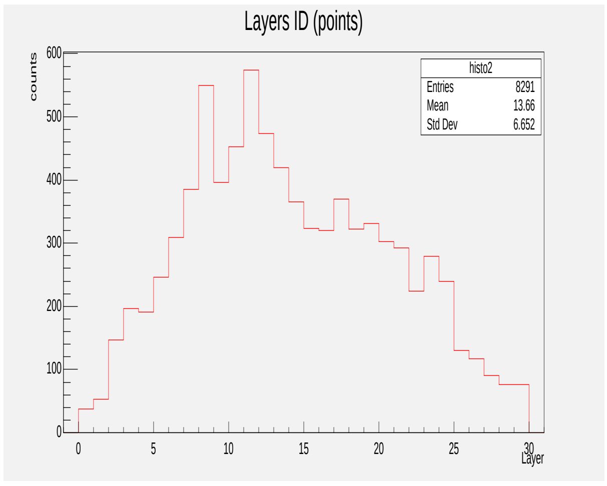


Figure 12: Points per layer

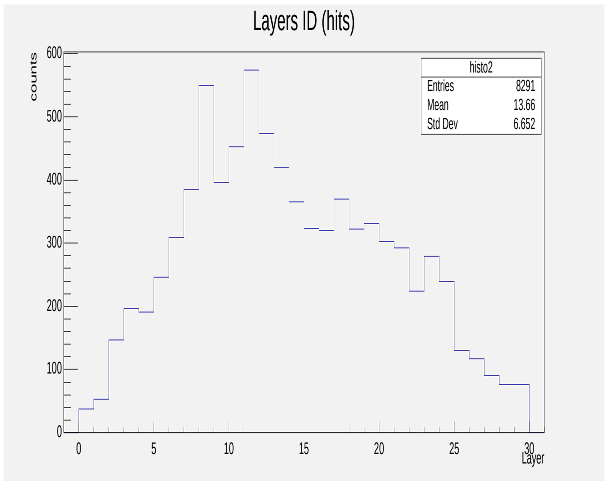


Figure 13: Hits per layer

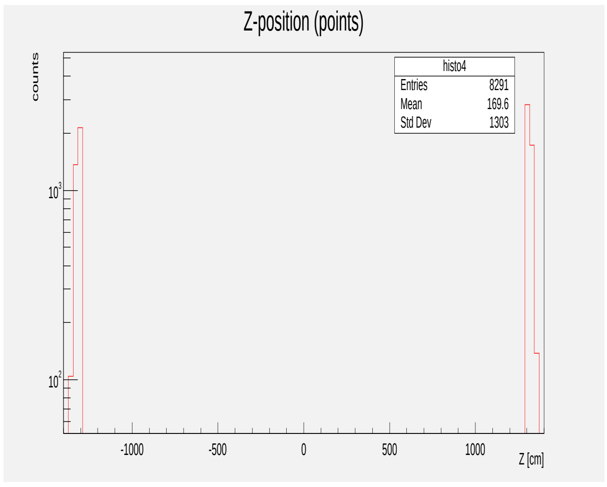


Figure 14: Points per Z-position

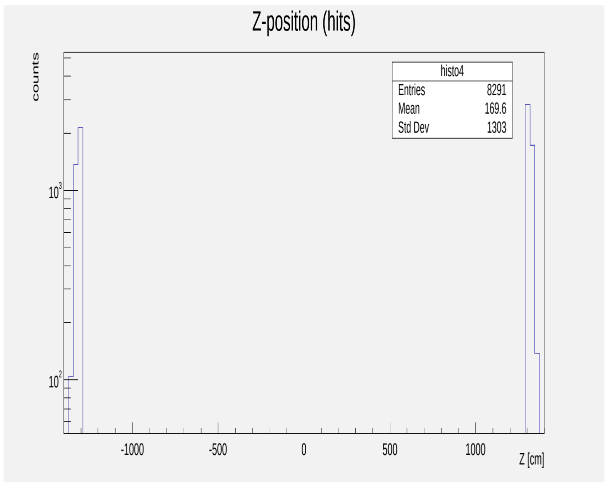


Figure 15: Hits per Z position

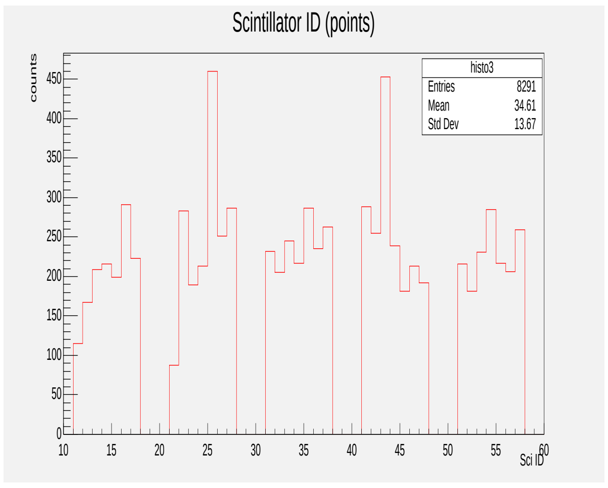


Figure 16: Number of points per scintillator copy number

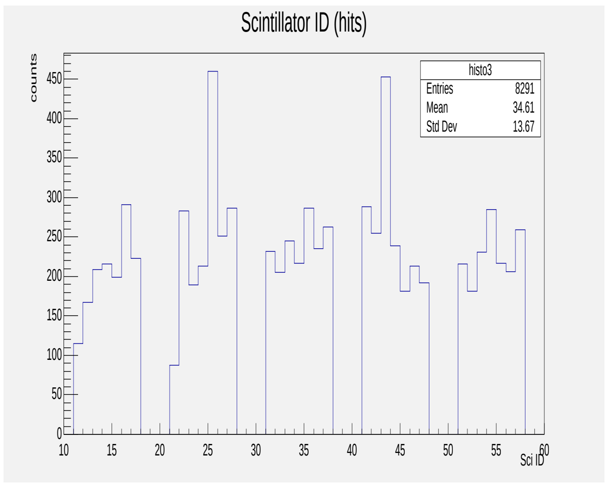


Figure 17: Number of hits per scintillator copy number

The graph shown in figure 18 was also obtained, displaying the values of the temporal resolution achieved through the method described in subsection 3.3.

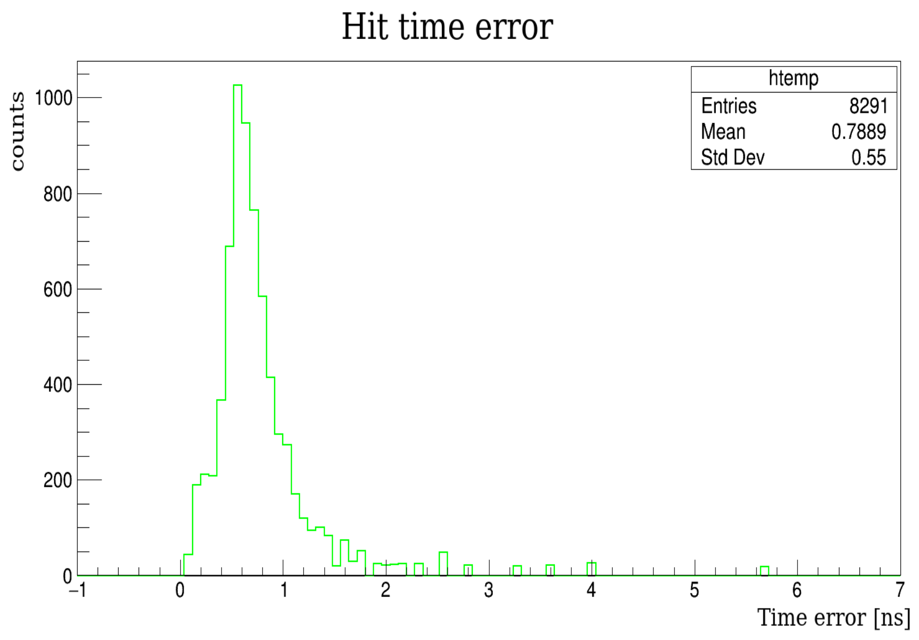


Figure 18: Hit time error

To analyze the relationship between the detector response energy (MCHit) and the energy collected in the simulation (MCTruth), the following graph was constructed.

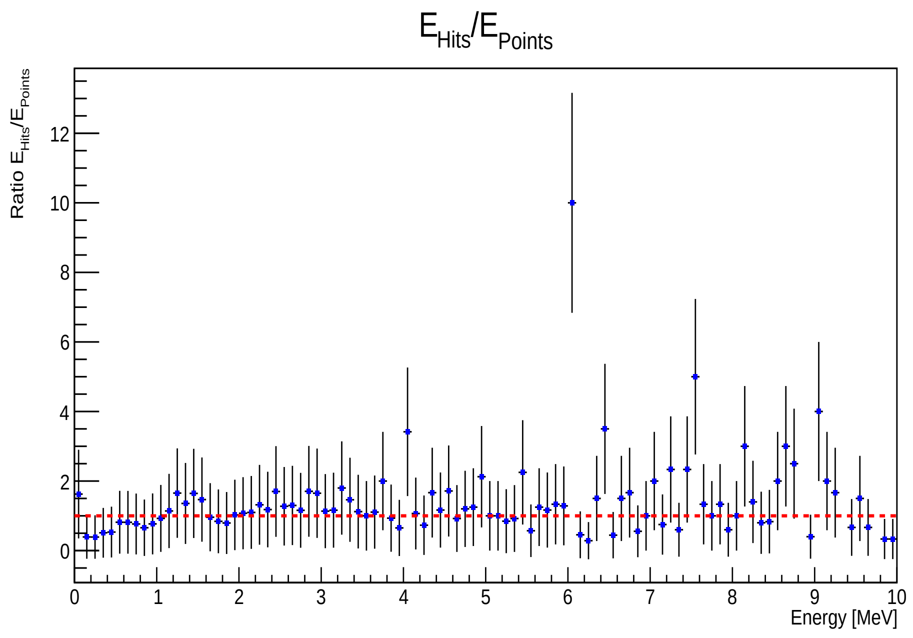


Figure 19: MCTruth-MCHit ratio

5 Conclusions

In this work, the ZDC has been successfully integrated into the SpdRoot framework, which is currently essential for the SPD experiment. The necessary classes for describing the geometry according to the materials and dimensions reported in [1] were implemented, along with methods to access these characteristics. In accordance with the new geometry design, the *points* classes and some of their getter methods were updated to access relevant information. Finally, digitization was performed using the currently reported ZDC parameters. The results presented in the figures and graphs of section 4 demonstrate the successful integration of the implemented code with the SpdRoot framework.

Future work includes implementing methods for spatial resolution and momentum reconstruction in our detector. Additionally, developing algorithms for clustering, track reconstruction in the ZDC, and particle identification remains as forthcoming tasks.

References

[1] *Abazov V.M. et al.* [SPD Collaboration] Technical design report of the Spin Physics Detector at NICA, 2024 *Nat. Sci. Rev.* **1** (1).

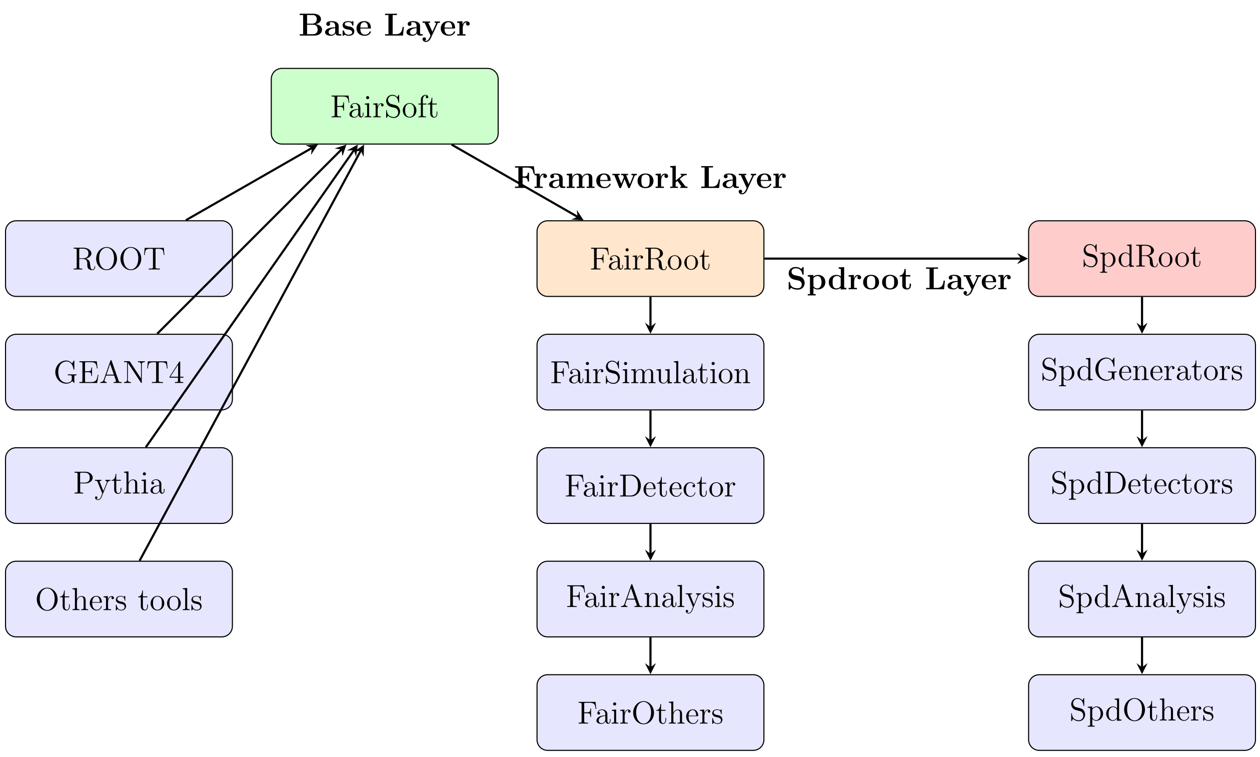
[2] *Pedraza M.* [Masther Thesis] Simulación del Calorímetro de Cero Grados (ZDC) en el SPD-NICA, 2025

[3] Offline Framework for the SPD experiment. https://git.jinr.ru/nica/spdroot

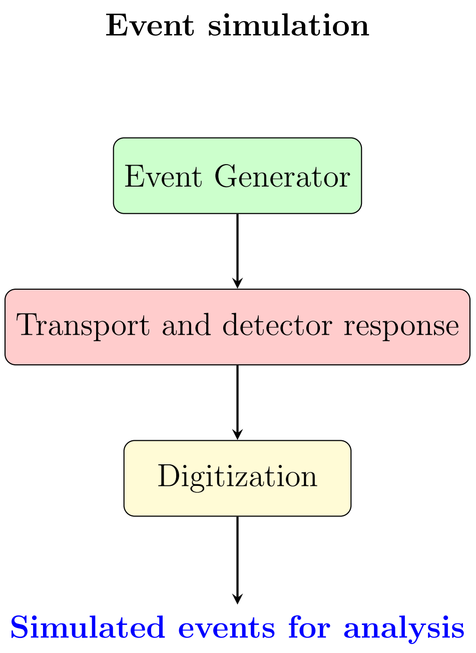
6 Acknowledgments

To my supervisor, Katherin Shtejer Díaz, for trusting me with this program and for her unwavering support throughout this work and my time as a Summer START student.

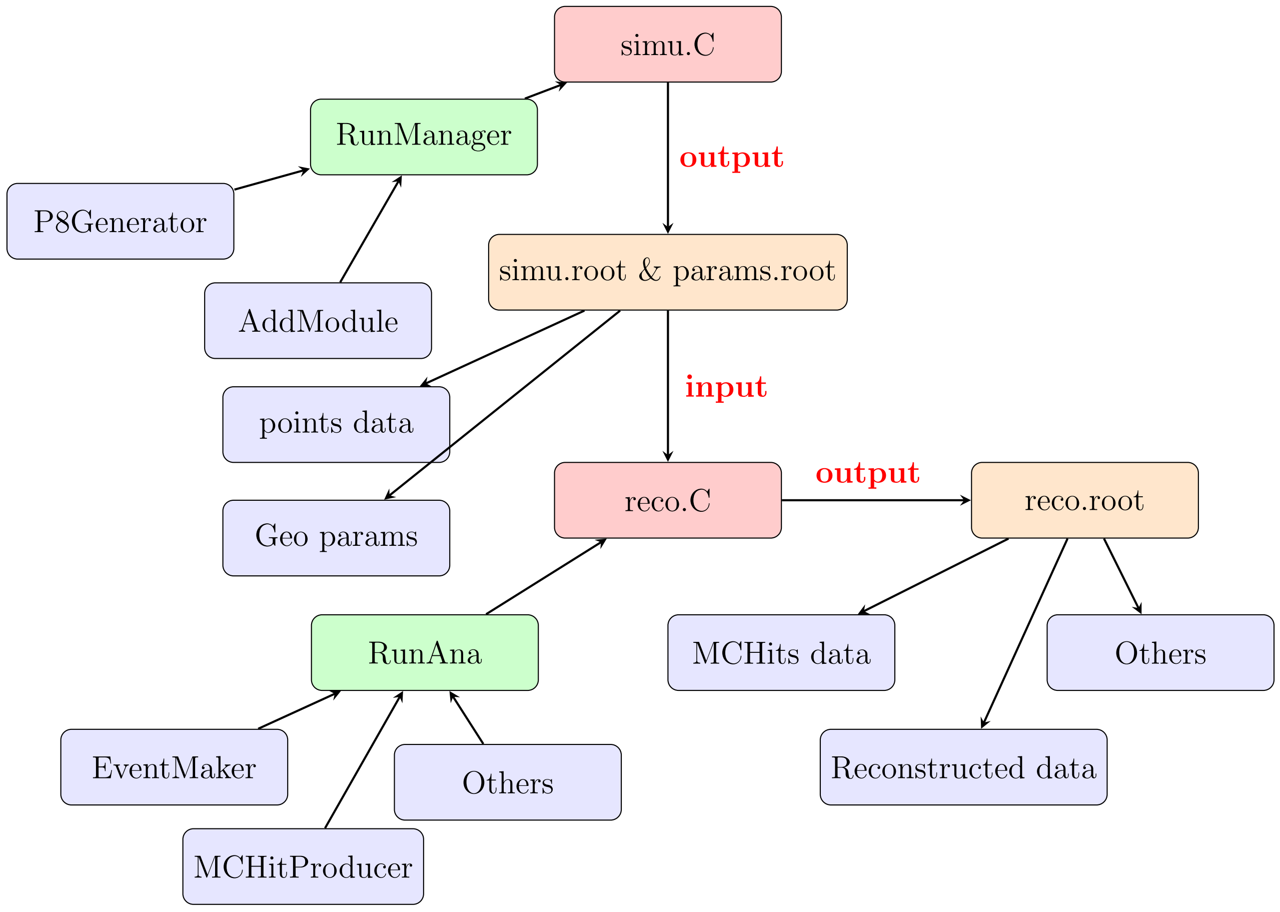
A Flowchart of the FairSoft/FairRoot/SpdRoot integration



B Algorithm for a simulation in SpdRoot (example with macros)


Event simulation
Event Generator
Transport and detector response
Digitization
Simulated events for analysis

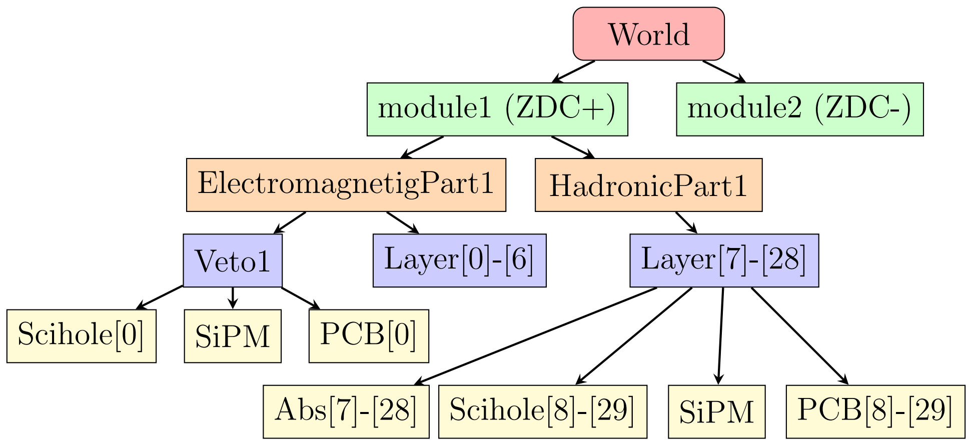
Macro example



C Creación de los punteros para los volúmenes lógicos

```
void SpdZdc::ConstructDetector()
{
//...
    TGeoVolume *Absorb[29]; //Pointer to 29 absorber logicals volumes
    TGeoVolume *PCB[30]; //Pointer to 30 PCBs
    TGeoVolume *Scihole[30]; //Pointer to 30 sci, + module
    TGeoVolume *Layer[29]; //Pointer later initialize as TGeoVolumeAssembly,
    +module
    TGeoCompositeShape *ScinShape[30]; //Pointer to 30 boolean shapes
    TGeoBBox *Sci[30]; //Pointer to 30 sci box
    TGeoBBox *SiPMbox=new TGeoBBox("SiPMbox",SiPMsize/2,SiPMsize/2,SiPMsize
   /2); //Solid SiPM volume
    TGeoVolume *SiPM = new TGeoVolume(mapper->AddPrefix("SiPM"),SiPMbox,med2
   ); //Logical SiPM volume
    Veto = new TGeoVolumeAssembly(mapper->AddPrefix("veto1"));//module+
    module1 = new TGeoVolumeAssembly(mapper->AddPrefix("module1"));//module+
    ElectromagnetigPart1 = new TGeoVolumeAssembly(mapper->AddPrefix("
   ElectromagnetigPart1"));
//...
}
```

D ZDC geometry hierarchy



E New methods for SpdZdcGeoMapper.h

```
class SpdZdcGeoMapper
{
//...
    Double_t GetModulexDim() const;
    Double_t GetElectrAbsThick() const;
    Double_t GetHadAbsThick() const;
    Double_t GetElectrSciThick() const;
    Double_t GetHadSciThick() const;
    Double_t GetSiPMThick() const;
    Double_t GetPCBThick() const;
//...
};
```

F Classes and methods for points

```
class SpdZdcPoint {
//...
    SpdZdcPoint(/*params*/, Int_t LayID /*New*/, Int_t SciID /*New*/,/*
   params*/);

    /*Two new methods*/
    inline Int_t    GetLayerId()       const { return fLayerId;}
    inline Int_t    GetSciId()         const {return fScintillatorId;}

    Int_t     fLayerId;          // if 0->veto ; else # Layer
    Int_t     fScintillatorId;   // ID (fScintillatorId/10->Row,
   fScintillatorId%10->Column)
//...
};
```

```
Bool_t SpdZdc::ProcessHits(FairVolume* vol)
{
//...
    fVolumePath = gMC->CurrentVolPath();

    SpdGeopathParser parser;
    parser.ParsePath(fVolumePath);

    fModuleID = parser.Num(2,true); //moduleID=> z+ is 1; z- is 2
    if (fVolumePath.Contains("veto1")||fVolumePath.Contains("veto2")) fLayerID=0; //Layer if ==0 ->Veto, else layer No
    else  fLayerID=parser.Num(4,true);
    fScintillatorID = parser.Num(5,true);

    AddHit();
//...
}
```

```
void SpdZdc::AddHit()
{
//...
    TClonesArray& clref = *fPointCollection;
    Int_t size = clref.GetEntriesFast();

    new(clref[size]) SpdZdcPoint(/*params*/,fLayerID,fScintillatorID,
                                 /*params*/);
}
```

G Hit production: classes and methods

```
class SpdZdcMCHit
{
//...
    void      SetLayId(Int_t Layid)            { fLayId = Layid;     } //Hit Layer Id
    void      SetTileId(Int_t tileid)          { fSciId = tileid;    } //Hit Sci Id
//...
    Int_t     GetLayId()        const { return fLayId;       } //Getter
    Int_t     GetTileId()       const { return fSciId;       } //Getter
//...
    Int_t       fLayId;         // layer id (0-29)
    Int_t       fSciId;         // Sci id (fSciId/10-> ColumnID) (fSciId%10-> RowID)
//...
};
```

```
Double_t SpdZdcMCHitProducer::CalibrationResEnergy(Double_t MCedp, Int_t LayerID )
{
    //ElectrCal
    TRandom3 rand(0); /*Random number*/
    Double_t edepMeV = MCedp*1.0e03; //MeV converssion
    if (LayerID<8) /*Electromagnetic section*/
    {
        Double_t RMS = ElectrStoch/sqrt(edepMeV);
        return std::max(0.0,rand.Gaus(edepMeV,RMS)); //Energy smear
    }
    else
    {
        Double_t RMS = HadStoch/sqrt(edepMeV);
        return std::max(0.0,rand.Gaus(edepMeV,RMS));
    }
}
//_____________________________________________________
```

```cpp
Int_t SpdZdcMCHitProducer::SmearPhe(Double_t Caledp,Int_t LayerID)
{
     TRandom3 rand(0); /*Random number*/
    if (LayerID<8) /*Electromagnetic section*/
    {
        Int_t NoPhe = (int)(ElectrYield*Caledp);
        Int_t NoPheSmear = std::max(0.0,rand.Gaus(NoPhe,sqrt(NoPhe))); //Phe
    smear

        return (int) NoPheSmear;
    }
    else
    {
        Int_t NoPhe = (int)(HadYield*Caledp);
        Int_t NoPheSmear =std::max(0.0,rand.Gaus(NoPhe,sqrt(NoPhe)));
        return (int) NoPheSmear;
    }
}
//_____________________________________________________
Double_t SpdZdcMCHitProducer::Signal(Int_t SmearedPhote,Int_t LayerID)
{
    if (LayerID<8) return (double) SmearedPhote/ElectrYield; //signal
   production
    else return (double) SmearedPhote/HadYield;

}
//_____________________________________________________
Double_t SpdZdcMCHitProducer::TimeRes (Double_t signal)
{
    return ResTimeStcoh/sqrt(signal);
}
//_____________________________________________________
void SpdZdcMCHitProducer::Exec(Option_t* opt)
{
//...
    for (Int_t i(0); i<npoints; i++)
   {
        point = dynamic_cast<SpdZdcPoint*>(fPointsArray->At(i));
        Double_t eLoss = point->GetEnergyLoss(); //method for get point edep
        if (!AcceptPoint(point) || eLoss==0)  //criteria for acceptance
        {
            nhfailed++;
            continue;
        }

   }
//...
    Int_t Layer = point->GetLayerId();
    Int_t TileNo = point->GetSciId();
    hit->SetLayId(Layer);
    hit->SetTileId(TileNo);

    Double_t Caledep = CalibrationResEnergy(eLoss,Layer);
    Int_t NoPhe = SmearPhe(Caledep,Layer);
    hit->SetResp(Signal(NoPhe,Layer));

    hit->SetHitTimeError(TimeRes(Signal(NoPhe,Layer)));
//...

}
```